Implementation of a Distributed Data Model and Format for High Performance Computing Applications

Jerry A. Clarke
U.S. Army Research Laboratory Major Shared Resource Center
Aberdeen Proving Ground , Maryland 21005-5067
Raytheon Company
clarke@arl.mil

Jennifer J. Hare
U.S. Army Research Laboratory Major Shared Resource Center
Aberdeen Proving Ground , Maryland 21005-5067
jen@arl.mil

Jerome D. Brown
U.S. Army Research Laboratory Major Shared Resource Center
Aberdeen Proving Ground , Maryland 21005-5067
High Performance Technologies Inc.
jerome@arl.mil

Abstract

In order to build reusable tools and software facilities for High Performance Computing (***HPC***) applications, the need for a common data model and format (***DMF***) is critical. Combining a DMF with a distributed access capability allows coarse grain coordination of multiple components during runtime. We have recently added such a distributed data hub to the "Distributed Interactive Computing Environment" (***DICE***).

Known as the DICE Object Directory, this hub incorporates Network Distributed Global Memory (***NDGM***), Hierarchical Data Format version 5 (***HDF5***) and the eXtensible Markup Language (***XML***) to provide a flexible yet efficient data exchange mechanism. Since the Heavy Data (large data arrays) is logically and potentially physically separate from the Light Data (data about the data, and small quantities of values), a significant amount of flexibility exists in the design of initialization, data access, and post-processing tools.

Introduction

The need for a common data model and format becomes clear when attempting to integrate two or more High Performance Computing (HPC) tools or codes into a single application. In the past, this has typically been done on a case-by-case basis. The resulting solution may well be suited for the particular application being developed, but may be difficult to extend to new applications.

Recently, there have been several major efforts to develop a robust data model and format for scientific computing. There are targeted efforts like NASA's Earth Observing System HDF-EOS project, and more general efforts targeted at a wider range of applications. The DOE ASCI Scientific Data Management (SDM) effort is one such example. If this effort is widely accepted, it will undoubtedly result in a variety of revolutionary tools for HPC. These efforts supply a comprehensive data model capable of servicing virtually any scientific HPC application. The development of tools to take full advantage of this data abstraction may require significant time to appear, however, and may need to sacrifice performance and/or ease-of-use to fully exploit the abstraction's capabilities. It seems reasonable that there will be a continuing need for a more targeted data model and format, at least for the near future, particularly if utilities and tools to exchange and visualize large datasets during runtime are included

The Distributed Interactive Computing Environment (DICE) is a collection of Mega-Components that provide HPC applications with Heterogeneous Distributed Shared Memory (DSM), Data Organization, Graphical User Interface (GUI) tools, and Scientific Visualization. Using these Mega-Components, User-Ready computing environments have been developed in support of several computational technology areas including: Structural Mechanics, Fluid Dynamics, and Computational Chemistry

A New Approach

DICE has recently implemented a data model and format, probably best described as a distributed data hub, called the DICE Object Directory. It is currently being integrated into several major HPC codes. The first major features of this distributed data hub is that the *Light Data* (data about the data and small amounts of values) is logically and, potentially, physically separate from the *Heavy Data* (large arrays). The second major feature is the support of Network Distributed Global Memory (***NDGM***) as a virtual file driver under NCSA's Hierarchical Data Format Version 5 (**HDF5**). This allows codes to efficiently access distributed data as if it were in a local disk file. HDF5 is currently being used in both the DOE and NASA efforts.

This data hub provides a targeted data model and format as well as a facility for sharing the data in a distributed environment during runtime. Through the use of Network Distributed Global Memory, a DSM-like facility for heterogeneous environments, codes and tools can synchronize their activity at a coarse grain level to provide a complex end-user application consisting of simple components. Access to the data hub is provided via system programming languages like C, C++, and FORTRAN, as well as scripting languages like Tcl/Tk (current), Python and Perl (future).

The primary advantage of such a data hub is interoperability. Tools can quickly be designed that perform a particular task very well. If they use a common data hub, these tools can then be reassembled in a variety of configurations to accomplish different goals. New HPC codes or visualization tools need only provide access to this data hub in order to use any of the other tools. This hub is both a data model and format; information about data values and "how they are to be used" are both available in the hub.

The design of this framework where HPC codes and tools can exchange information at runtime is a balancing act between functionality, efficiency, and ease of integration. Vendor specific solutions, while quite efficient, may be inherently non-portable. Object-oriented computing infrastructures, while portable and flexible, may be difficult to integrate with existing codes.

The data format provides the specification of how data values are stored, while the data model provides information on how they are intended to be used. Several key components were identified as crucial to the acceptance of the system:

1. A portable yet efficient format for storing large amounts of data values.
2. A portable, flexible model for describing the data content.
3. An efficient method for exchanging large amounts of data between processes that are potentially executing on separate hosts.
4. A flexible way to glue together large, powerful software components.
5. Portable Graphical User Interface components
6. Visualization Support

A major concern is that the system is based on components that become unavailable, unsupported or too expensive. Using components that are industry standard and available on HPC resources for the foreseeable future will be mitigates this risk.

Data Format

The data format, used to store large amounts of data values, must be capable of storing character, integer, and floating point values in a portable fashion. It must be straightforward to convert between a variety of data precision and host dependent formats. In addition, since the layout of the data is tightly related to access efficiency, it is also quite beneficial if multi-dimensional data access facilities are included as a part of the interface.

The Hierarchical Data Format Version 4 has undergone a total re-write, mainly to support the DOE ASCI and NASA EOS effort. In addition to well-conceived access to a variety of data types, HDF Version 5 includes multi-dimensional data access support and a "virtual file driver" interface. With the virtual file driver interface it is possible to allow transparent access, via the HDF5 API, to a variety of low level data storage systems.

HDF5 also includes a data compression facility so that data can be compressed and decompressed as is migrates to secondary storage. All of these features make HDF5 a good choice for storing large amounts of vales through a straightforward data abstraction at a reasonable level of efficiency.

Data Model

Since it is anticipated that any model will need to be augmented by the application in order to include all of the necessary information, flexibility and extensibility are key elements to the data model. The intent of the data model component is to provide a way to easily describe data content as opposed to data value.

This is quite similar, in concept, to a Web page where the content of the information is separate from the display of the information (i.e. Text color is independent of a word's meaning). To assist in the free exchange of data on the internet, the "World Wide Web Consortium" (W3C), an organization whose members include AOL, IBM, Microsoft, Oracle, Sun, and other major corporations, proposed a standard known as "eXtensible Markup Language" (XML). XML is pervasive on the Web and is supported by a vast number of tools both free and commercial.

The base data model including information like grid topology, scalar names, and physical data location, is stored using XML. This differs from other DMF efforts, like the DOE effort, where all information is stored in the HDF5 format. The ability to separate "light" data from the "heavy" data values, however, allows many tasks (e.g. setup) to be performed without accessing the heavy data values at all. In addition, this makes it easy for separate components to view the same data values differently. For example, one component's structured grid may be viewed by another component as a collection of hexahedra. We believe that the ability to physically separate the light data from the heavy data provides an enormous benefit.

Data Exchange

"Network Distributed Global Memory" is currently in use as a data exchange mechanism for use in runtime visualization efforts, and it's performance characteristics are well known [(1)]. By directly writing to system shared

memory when possible, NDGM can efficiently transfer large amounts of data between HPC applications. For distributed applications where shared memory transfers are not possible, NDGM uses an efficient low level socket layer that can take advantage of vendor specific interface enhancements. In addition, NDGM views data as a contiguous buffer of bytes. This makes the development of an HDF5 virtual file driver straightforward.

The use of more standard, object oriented, data exchange facilities like CORBA may be included in the future. Other software, distributed shared memory systems, like TreadMarks, use memory-paging algorithms to aid in the management of data exchange. Currently we have no plans to implement other such facilities into this system due to performance concerns.

Application Glue

The components of a large system are generally developed using system-programming languages like C, C++, or FORTRAN. Once these computationally intensive components have been developed however, they may be "glued" together in a number of ways to provide the overall required functionality. Using a system-programming language for this task is tedious, time-consuming, and inflexible.

Scripting languages are specifically designed for this purpose. They tend to be "weakly-typed" so that the output of one component can easily be used as the input to another with little concern over the "type" of the data. While one pays in runtime efficiency for this flexibility, scripting languages are intended to call large chunks of functionality and are not used for fine grain control.

While significantly different in syntax, most scripting languages are similar in functionality. The choice of a scripting language can regularly be a matter of personal preference. We have chosen to implement most scripting tools using the Tool Command Language (**Tcl**). The loose typing of variables, tight integration with graphical user interface facilities, and the corporate level support on a variety of platforms make it a stable platform for upper level development.

One can extend the functionality of any of these scripting languages by adding additional commands. In addition to developing callable functions in the language itself, this is accomplished by adding a "wrapper" to interface the language command calling interface with the argument list of a system-programming routine that implements the desired functionality efficiently. "*The Simplified Wrapper and Interface Generator*" (**SWIG**) is a widely used tool that generates these wrappers for code written in ' C' or C++ for access via languages like Tcl, Python, Perl, and Java. SWIG allows commands to be added to these languages that access the custom code in an object-oriented fashion. We use this tool extensively to provide a consistent programming structure over several very different programming languages.

Graphical User Interface

Tcl is tightly integrated with a graphical user interface widget set called Tk. Most scripting languages provide access to Tk which in turn implements widgets like labels, buttons, sliders, and text input boxes. Since Tcl/Tk is available on every major platform, graphical user interfaces and tools written with this combination are extremely portable.

The addition of frequently used mega-widgets, like file selection boxes, is accomplished via an additional, pure Tcl/Tk toolkit called BWidgets. Since this toolkit requires no platform compilation, portability issues are reduced. The increased efficiency of modern scripting languages, together with the typical power of a modern desktop workstation, make pure scripting toolkits fast enough for most graphical user interface needs.

Visualzation

There are a variety of excellent commercially available scientific visualization systems. These include AVS, EnSight, IDL, and PV~Wave. Recently, however, the "Visualization Toolkit" (vtk) has emerged as one of the most popular, freely available, visualization system. Vtk is a large C++ class library of visualization algorithms and graphics routines that can be accessed from system programming languages as well as Tcl, Python, and Java.

Through extensive support from the medical imaging community and the DOE, vtk continues to gain users and contributed support. While vtk may not be the most computationally efficient choice, the functionality provided is unparalleled. Internal visualization in DICE is handled exclusively though vtk networks. Facilities for visualizing computational grids, cutting planes, isosurfaces, vector fields, and volume visualization are all included as part of the tool set. In addition, we have developed a data reader for the EnSight commercial visualization package. The combination of support for vtk and EnSight serves the vast majority of user needs.

Assembling a Total System

Using the distributed data hub and other components available in DICE, a total end user application system can be assembled. This system provides access to pre-processing, code setup, runtime support and post processing.

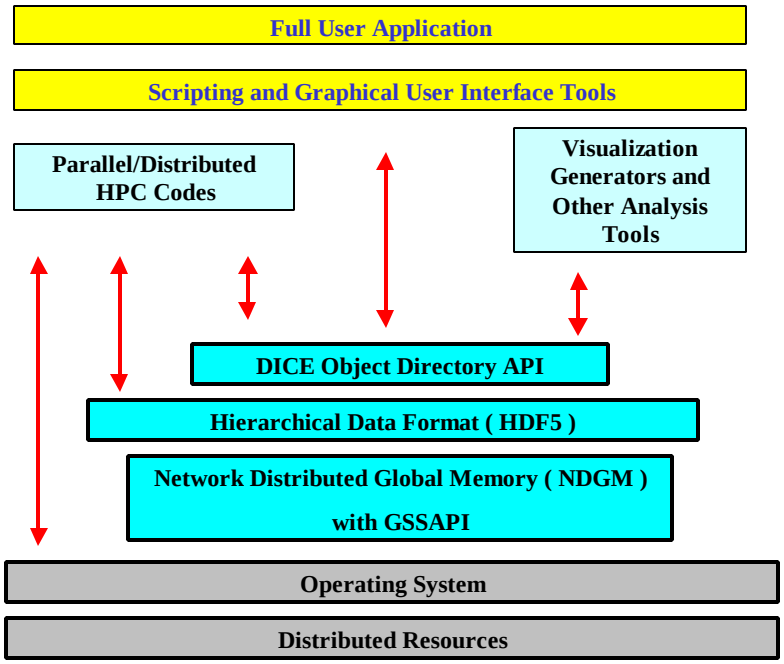


Figure 1. Access to the DICE Object Directory

As shown in figure 1, Network Distributed Global Memory provides efficient access to a virtual, contiguous buffer via a client/server architecture. NDGM servers manage a portion of a buffer that physically resides in system shared memory, a disk file, or in it' s local address space. Clients access this buffer via a puts(), gets(), and vector operations by specifying a virtual start address of the desired data. Locating sections of requested buffers and the actual transfer is handled by the NDGM client system. In addition, NDGM provides semaphores and barriers to coordinate the activity of parallel/distributed applications.

While NDGM is not a true DSM system it provides several advantages when designing an efficient data hub. NDGM does not automatically map and unmap pages of memory from an application. Rather, the application itself is responsible for transferring data to NDGM via a read/write interface. While this at first might appear as a major inconvenience, it greatly reduces the amount of communication. In addition, NDGM takes full advantage of operating system shared memory facilities when server and client reside on the same machine. This allows remote clients to maintain full access to the data buffer while local clients incur minimal overhead. Support for the "Generic

Security Services API" (**GSSAPI**) has been added to NDGM in order to provide the necessary security features that are typically required in an HPC environment.

HDF5 is used to provide the NDGM buffer with structure. HDF has been enhanced to support data in NDGM via the "virtual file driver" interface. So now an HDF dataset can reside on disk, in NDGM, or in both. This is particularly useful when data has both a static and dynamic component. For example, a static grid may reside on disk while the updating solution resides in NDGM. HDF provides a consistent interface for structured and unstructured data as well as a grouping structure.

While HDF provides a full featured attribute facility capable of storing Light Data such as units and dimension names, we feel that a better choice is to take advantage of the recent emergence of eXtensible Markup Language (XML). Although primarily targeted at web based applications, XML provides a standard way to store and structure application specific data. There is already an impressive array of tools available for parsing XML and converting it to internal data structures. By utilizing the functionality of XML, and logically separating the Light Data from the HPC Heavy Data, a myriad of tools can be built with high level scripting languages and web tools that allow intelligent queries of enormous datasets without causing massive amounts of I/O activity.

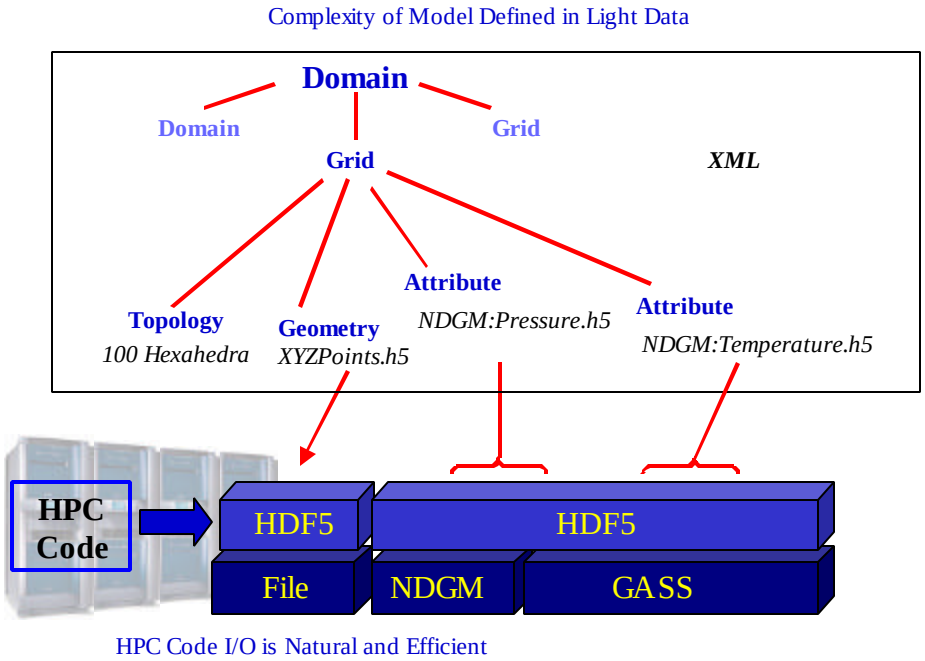


Figure 2. Separating Light and Heavy Data

The concept of separating the Light Data from the Heavy Data**,** as shown in figure 2, is critical to the performance of this data model and format. HPC codes can read and write data in large, contiguous chunks that are natural to their internal data storage to achieve optimal I/O performance. If codes were required to significantly re-arrange data prior to I/O operations, data locality, and thus performance, could be adversely affected, particularly on codes that attempt to make the maximum use of memory cache. The complexity of the dataset is described in the Light Data portion that is small and transportable. For example, the Light Data might specify a topology of one million tetrahedra while the Heavy Data would contain the geometric XYZ values of the mesh and pressure values at the cell centers stored in large, contiguous arrays. This key feature will allow reusable tools to be built that do not put onerous requirements on HPC codes.

In addition to accessing data values directly, via the HDF interface, the DICE Object Directory contains an object-oriented convenience interface to the XML Light Data and HDF5 Heavy Data. This convenience interface layer provides access to a subset of the HDF calls using reasonable default values for many of the parameters. It is also able to encapsulate and synchronize data, in memory and external (HDF) representations, to assist in the

development of a more interactive set of data analysis tools. Through the use of SWIG, this interface is available for common scripting languages like Tcl, Python, and Perl. This allows the rapid development of a wide range of utilities for both importing and exporting data to common HPC formats. In addition, any package that provides access to this interface layer will automatically inherit runtime capabilities since the data maintains its distributed characteristics via NDGM.

Heavy Data

The Heavy data is a subset of the HDF5 data model. Currently only homogeneous arrays are supported at all levels. Compound data types, used to implement data structures, must be separated into homogeneous, multi-dimensional arrays. This not only aids in simplifying upper layers of the design, it also provides performance benefits.

Just as in HDF5, data is considered to have both ***Type*** and ***Space***. Type is the number type of the base datum (integer or floating point) along with it' s precision and byte order. Space describes both the shape (rank and dimensions) of the dataset and a sub-region selection. Since the end goal is to be able to efficiently transfer data in a distributed environment, type and space are defined for both the source and destination of a data transfer. In this fashion, a sub-region of a IEEE 32 bit floating point array can be efficiently transferred to different region of a little-endian 64 bit floating point array by properly specifying the type and space of both sides.

The DICE Object Directory supports 8, 32, and 64 bit integers and floating point data types. Each data type has a corresponding Array. Arrays are self-allocating, single dimensional, data structures. Arrays have methods to set and get the number of elements, safely access individual element values, and to directly manipulate the underlying data pointer for maximum efficiency. Several arithmetic operators have been overloaded and an additional "expression" facility has been supplied to allow operations on entire arrays. Perhaps the best way to describe arrays is with an example. Below is a portion of C++ code to create and manipulate a 32 bit floating point array

```
DiceFloat32Array    *MyData = new DiceFloat32Array;          // Create a new array
MyData->SetNumberOfElements( 100 );     // Allocate space for 100 elements
MyData->Generate( 10.0, 20.0 );          //  Initialize the entire array with vaules
                                         //  10.0, 10.01, …20.0
MyData->SetValue( 0, 3.14 );             // Set the first value
MyData += 100.0;                         //  Add 100 to every element
memcpy(  MyData->GetVoidPointer(),
                OtherData,
                100 * sizeof( float ));  // Blindly copy data to pointer … Be Careful !
```

Arrays are made multi-dimensional by associating a space with the array. A space has rank (the number of dimensions) and dimensions (the length in each direction). A sub-region of this space may be described by either Hyperslab or Index. A Hyperslab specifies a Start, Stride, and Count in each dimension while an Index describes absolute indexes into a dataset. Commonly, Hyperslabs are used to subsection structured datasets while Indexes are used to subsection unstructured datasets. As an example, the following code provides a full space description of an array :

```
DiceFloat64Array        *MyData = new DiceFloat64Array;
DiceSlabSpace           *MySpace = new DiceSlabSpace;

MyData->SetNumberOfElements( 2000 );
MySpace->SetRank( 3 );                   // A 3D Array
MySpace->SetDimension( 0, 100 );
MySpace->SetDimension( 1, 10 );
MySpace->SetDimension(2, 2 );            // That is 100 x 10 x 2 = 2000
MySpace->SetSlabToAll();                 // Select the entire array
MySpace->SetStart( 0, 10 );
```

```
MySpace->SetStride( 0, 2 );
MySpace->SetCount( 0, 5 );          // From the first dimension (100)
                                    // select only elements 10, 12, 14, 16, and 18
                                    // the total selected size is 5x10x2 = 100
```

Similarly, selections may be made by absolute index into the array. This is accomplished by providing an array as an argument to a method of DiceIndexSpace. Each element in this array is an absolute offset into the defined space. the DICE Object Directory does not allow for boolean operations on spaces (i.e. the union or intersection of two spaces) or the mixing of Hyperslabs and Index in the same space.

Data type, space, and an arrays are associated by the construction of a "container object" called DiceMemory. This merely allows for a smaller argument list on many methods and simplified bookkeeping for the final application.

All Heavy data is stored in HDF5 "files". HDF provides a "Virtual File Driver" layer to allow HDF5 "files" to physically reside in things other that standard disk files. The DICE Object Directory provides a driver NDGM in addition to the available Global Access to Secondary Storage (**GASS** : Globus System) and CORE (In-Memory) drivers provided with HDF5.

HDF5 files in the DICE Object Directory are organized similar to a UNIX filesystem. While HDF5 provides a full Directed Acyclic Graph (DAG) structure, the DICE Object Directory limits this to a strict hierarchy (i.e. no links). Datasets inside an HDF5 files must be fully qualified for access. This is done by providing the Domain, File, and Pathname of the dataset. This is passed as a colon separated string (i.e. NDGM:Myfile.h5:/Geometry/XYZdata). Accepted domains are : FILE, CORE, GASS, and NDGM. To access Heavy data in the DICE Object Directory, use the DiceHDF object :

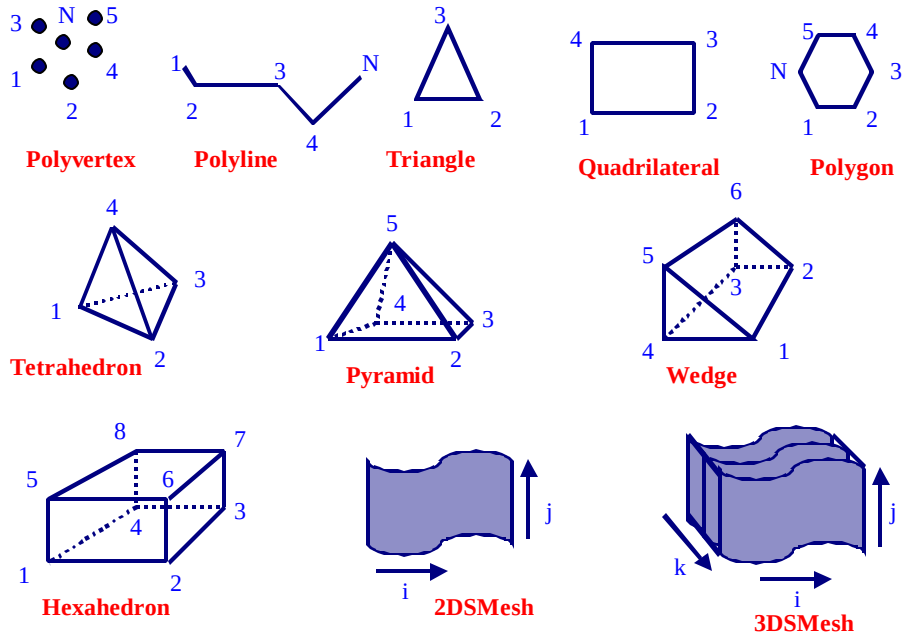
```
DiceHDF          *MyH5File = new DiceHDF;

MyH5File->SetAccess ("rw");
MyH5File->Open("NDGM:Grid.h5:/Block 1/Geometry/XYZ" );
MyH5File->Read( MyDiceMemory );
MyH5File->Close( );
```

Light Data

The Light data in the DICE Object Directory is primarily used to represent the data model. This provides the knowledge of what is represented by the Heavy data. In this model, HPC data is viewed as a hierarchy of ***Domains***. A Domain may contain one or more sub-domains but must contain at least one ***Grid***. A Grid is the basic representation of both the geometric and computed/measured values. A Grid is considered to be a group of elements with homogeneous ***Topology*** and their associated values. If there is more than one type of topology, they are represented in separate Grids. In addition to the topology of the Grid, ***Geometry,*** specifying the X, Y, and Z positions of the Grid is required. Finally, a Grid may have one or more ***Attributes***. Attributes are used to store any other value associated with the grid and may be referenced to the Grid or to individual cells that comprise the Grid.

We have chosen an initial set of base topologies to support. Since triangles and quadrilaterals are widely used in a variety of codes, it seemed reasonable to directly support them in addition to the generic polygon. In several of the topologies, there is potentially more than one reasonable way to order the vertices. For this reason, the DICE Object Directory specifies a default ordering. It is possible to override this ordering. While facilities to extend this list are included, this subset provides an excellent start.



DICE Object Directory Base Topologies

Figure 3. DICE Object Director Base Topologies

As mentioned earlier, the data model is stored externally in an XML file. XML looks similar to HTML except that the "elements" are specific to the application. In the case of the DICE Object Directory, the elements are used to express the data model. Below is an example of the DICE Object Directory XML to express 200 tetrahedra with pressure values specified at the center of each cell :

```
<?xml version="1.0" ?>
<!DOCTYPE dice SYSTEM "dice.dtd" >
<Dice>
<Dod>
<Domain>
        <Grid Name="My Unstructured Grid with Pressure at Cell Centers">
                <Topology
                Type="Tetrahedra"
                NumElements="200">
                </Topology>
                <Geometry>
                        <Vertices>
                                <Index
                                Location="HDF"
                                Type="List">
                                MyData.h5:/Geometry/Connections
                                </Index>
                                <Points
                                Loacation="HDF"
                                Type="XYZ">
                                MyData.h5:/Geometry/XYZ
                                </Points>
                        </Vertices>
                </Geometry>
                <Attribute Name="Pressure">
```

```
                    <Values
                    Type="Scalar"
                    Center="Cell">
                    MyDaya.h5:/Scalars/Pressure
                    </Values>
                </Attribute>
        </Grid>
</Domain>
</Dod>
</Dice>
```

This example is a typical situation where the XYZ points are stored in an array. The "connection list" is an index into the points array. In this case there are 4 connections specified for each tetrahedron. So there are 200 x 4 = 800 values specified in the file Mydata.h5 under the dataset name "Connections" in the HDF5 group "Geometry". Geometry may be a simple or as complex as required. The most complex situation would be if "points" were indexed to created "vertices" which were indexed to create "edges", which were indexed to create "faces" which were indexed to create "volumes".

The XML used to describe the data model in the DICE Object Directory is typically generated by an HPC code or a script. The XML is used by other applications, like the DICE Visualizer, to initialize any necessary internal structures needed to handle the data. There are convenience objects in the DICE Object Directory Application Programmers Interface to help with the parsing of the XML. These convenience objects are roughly based on the W3C Document Object Model (DOM) specification, but take liberties with both the interface names and operation to optimize functionality and performance.

The objects **DiceGrid** and **DiceAttribute** can parse/serialize the XML and access the heavy data in HDF5 as described in the XML. Parsing involves reading the XML text and creating a hierarchical tree structure, in memory, to represent its' contents. Methods are provided to navigate this tree structure and retrieve or modify its' contents. Serializing XML is the reverse operation ; the internal tree structure is converted into a valid text representation. Applications use these two objects to transfer data in and out of the distributed data hub. The base object of both DiceGrid and DiceAttribute is the **DiceDom** object. It may be used independently to obtain information about the XML if Heavy data access is not required.

All light data and external control is expressed in DICE via XML. This allows parsers to be consistent across a wide variety of tools without concern for the transport mechanism of the data. The XML may be passed as an argument, stored in an external file or communicated via a socket mechanism. For customization purposes, tools can also augment the standard content with XML "processing instructions". This is useful for attaching peer level information to the standard XML content without modifying the base specification.

Conclusion

The DICE Object Directory and accompanying DICE tools are intended to provide many of the necessary functions required to develop an entire operating environment for an HPC code. This could be a full interface to one particular HPC code or part of a full-featured Problem Solving Environment. It is not intended that this distributed data hub support all possible data needs. By providing vertical support for a targeted class of HPC codes, we hope to extend the lifetime, portability, and utility of these codes while at the same time assisting the computer science community in the development of data models and formats as they gain acceptance and support.

References

1. Jerry Clarke, "Emulating Shared Memory to Simplify Distributed-Memory Programming", IEEE Computational Science & Engineering, Vol4, No. 1, pp 55-62, January-March 1997

2. See the listing under National Center for Supercomputing Applications, "HDF5 – A New Generation of HDF", http://hdf.ncsa.uiuc.edu/HDF5 (current April 21, 2000)

3. See the listing under World wide Web Consortium, "Document Object Model (DOM) Level 1 Specification", http://www/w3.org/TR/REC-DOM-Level-1 (current April 21, 2000)

4. See the listing under Scriptics, "Scripting: Higher Level Programming for the 21st Century", John Ousterhout, http://www.scriptics.com/people/john.ousterhout/scripting.html (current April 21, 2000)

5. Will Schroeder, Ken Martin, Bill Lorensen, "The Visualization Toolkit – 2nd edition", Prentice Hall PTR, Upper Saddle River, NJ, 1998

6. See the listing under Los Alamos National Laboratory, "Scientific Data Management", http://www-xdiv.lanl.gov/XCI/PROJECTS/SDM/index.html (current April 21, 2000)

7. See the listing under NASA, "HDF-EOS Standards and Tools Information Center", http://hdfeos.gsfc.nasa/hdfeos/workshop.html (current April 21, 2000)

8. See the listing under National Center for Supercomputing Applications, "A Survey of Visualization Tools for High Performance Computing, SIAM Parallel Processing for Scientific Computing March 1999", Randy Heiland, http://www.ncsa.uiuc.edu/~heiland/SIAMPP99 (current April 21, 2000)

9. See the listing under Rice University, "The TreadMarks Distributed Shared Memory (DSM) System", http://www.cs.rice.edu/~willy/treadMarks/overview.html (current April 25, 2000)

10. Jerry A. Clarke, Charles E. Schmitt, Jennifer J. Hare, "Developing a Full Featured application from an Existing Code Using the Distributed Interactive Computing Environment", DoD HPCMP Users Group Conference Proceedings, June 1998